```
0080       STA POINTL
0090       LDA #00
0100       STA POINTH
0110       JMP START
0120 .END
*ET
```

You may edit and re-edit the text until you're satisfied that it is correct. Notice that it isn't necessary to enter the lines in line number order, but they will be entered into memory and assembled in line number order. It is important to remember that you must have at least one space separating the fields in each line of text, that each line ends with a carriage return, and that each line must begin with a line number.

Now that you've had an opportunity to see the basic operation of the editor, we will examine each editor capability in detail. Table 1 shows the editor command summary, explaining briefly what each command does.

Table 1

EDITOR COMMAND SUMMARY

P A	Print all stored text
P n	Print text beginning at line n
F xyz	Find and print all lines containing string xyz
S	Print status - hex origin, end of text, and decimal number of lines in the current text file
R	Resequence line numbers by 5's
Q	Query - return to BASE= query
E	Exit to APPLE monitor
?	Set "?" as prompt character
?x	Set x as prompt character
A	Go to Assembler with current text file
K^C	Control K - exit to APPLE monitor
T^C	Control T - Save current text file on audio tape

BASIC EDITOR COMMANDS

Entering Text

Text is entered into memory by typing in the line of text preceded by a line number. Line numbers may range from 1 to 9999. There is no necessity to type leading zeroes, the editor will insert them automatically. Note that any line typed to the editor which does not contain a line number will be interpreted as an editor command. If the editor cannot recognize the line as a command, the error message BAD COM (bad command) will be printed on the screen.

It is not necessary to "format" the text by entering spaces. One space must separate each field on a line from the rest of the fields on that line, but you don't need to use the number of spaces required to make the text "line up" as it will after it is assembled. The assembler does that for you.

Correcting Text During Entry

If you type in a portion of a text line and realize you have made a mistake, there are two methods of correcting the error:

1. Depress the control key and hit the "X" key (control X). The line will be ignored and a carriage return/line feed will be issued by the editor.
2. Backspace by pressing the back arrow key (←), then retype the offending character. The editor will back up each time you press the back arrow key, moving one space to the left each time you press the key. When you have retyped the offending character, type the rest of the line - do not press the return key until you've reached the end of the entire line. All characters on the line will be deleted, beginning with the character

under the cursor and continuing to the end of the line, as soon as you press the return key. The deleted characters will not be removed from the screen, however. Example 5 shows the results of backspacing to correct an error.

Example 5

USING ← TO CORRECT ERRORS

step 1 10 THIS IS A MESPILLING
now the cursor is positioned immediately following the G
step 2 Press the ← key 8 times, and type ISPILLING
now the line of test looks like this:
10 THIS IS A MEISPILLING
and the cursor is still positioned after the G
step 3 Press the ← key 10 times and type ISPELLING
now the line looks like this:
10 THIS IS A MISPELLINGG
and the cursor is positioned over the last G on the line. Press the return key at this point. The offending G is not removed from the screen.
step 4 P A
The editor prints the line on the screen
0010 THIS IS A MISPELLING

Now that you have pressed the return key, the only way to correct the line is to retype it:

10 THIS IS A MISSPELLING

While you may well forget to continue typing the line after backspacing to correct an error (at least at first), it won't take long to get to the point where you do it automatically, since it works very much like the APPLE's own editor. Note that you cannot forward space using the ARESCO Text Editor, however.

Listing The Text

You've seen how to list the entire text file by typing P A. If you wish to only list a portion of the text, type P, a space, and a line number. The editor will then print all of the text, beginning with that line number, and continuing through to the end of the file.

If you wish to only list a few lines of text, type P, space, and a line number. The editor will begin printing out the text, beginning with that line number. When you have seen all the text you wish to see, press any key. This will signal the editor that you wish it to stop listing, and it will stop printing the text and wait for another command to be entered.

Note that depressing any key while the assembler is running will also stop the assembler listing. However, you will be returned to the editor, not to the assembler.

Adding A New Line To The Text

Each line you type will automatically be inserted in the text file in line number sequence. Thus, if you wish to add a new line between old lines 20 and 30, simply give the new line a number between 21 and 29. If you wish to insert a new line between two existing lines with adjacent numbers, you must first resequence the line numbers (see Resequencing Lines).

There is no restriction on the number of text lines or the size of the text file (except that line numbers must be in the range 1 - 9999). The only restriction, then, is the amount of memory available for such storage.

Resequencing Lines

The assembler ignores the line numbers in your file when doing an assembly. Thus, the line numbers are a convenience for you when you are editing text. If you wish to resequence the line numbers, simply type an R and a carriage return. The editor will automatically resequence all the line numbers in your file. The first line will be the line number 5 and each line number which follows will be incremented by 5.

Example 5

RESEQUENCING LINES

```
1 LINE 1
2 LINE 2
3 LINE 3
R
P A
0005 LINE 1
0010 LINE 2
0015 LINE 3
*ET
```

Locating Specific Characters In The Text

In a lengthy text file it is often desirable to be able to find all lines in the text which contain certain characters or groups of characters. This is done with the Find command. Example 6 illustrates the use of the Find command. First the entire text file is listed using the P A command, then all lines which contain references to POINTH are printed by issuing the

command F POINTH. The second portion of the example shows finding all lines which contain an asterisk by typing F *. You must always type a space after the F.

Example 6
FINDING SPECIFIED TEXT

```
P A
0010 POINTH = $FA
0020 POINTH = $FB
0025   *=$0000
0030 VAL1 = *+1
0040 VAL2 = *+1
0050 PROG CLC
0060      LDA VAL1
0070      ADC VAL2
0080      STA POINTL
0090      LDA #00
0100      STA POINTH
0110      JMP START
0120 .END
*ET

F POINTH
0020 POINTH = $FB
0100      STA POINTH
*ET

F *
0025   *=$0000
0030 VAL1 = *+1
0040 VAL2 = *+1
*ET
```

Exiting From The Editor

The user may return to the APPLE monitor by typing an E (Exit) command. Never exit the Assembler/Editor by pressing the RESET key. If your system uses the Apple Disk, and you press RESET while using the Assembler/Editor, you will have to reboot the DOS. You may return to the monitor using K^C (control K) if you wish.

OTHER COMMANDS

The Status Command

Typing an S command to the editor will result in the editor printing out three numbers. The first number is the hexadecimal starting address of the current text file. The second number is the hexadecimal address of the end of the current text file. The third number is the decimal number of lines contained in the current text file. If you are entering a text file which you suspect may approach the capacity of your available memory, you can check the amount of memory being used from time to time as you enter the text. You should note, however, that the S command will return incorrect information if no text has been entered into your current text file.

The Assemble Command

Typing an A command to the editor will terminate editor function and transfer control to the assembler program. The editor automatically configures the assembler for a memory-to-memory assembly of the current text file. You should note that the transfer is made to the assembler cold start entry point so this command may not be used in multiple file assembly. For the second and succeding files of a multiple file assembly, you must exit from the editor to the monitor and then enter the resident assembler at its warm start point. See the assembler documentation for further details.

Saving the Text on Audio Cassette Tape

To transfer your edited text to audio cassette for storage, first insert a cassette in your recorder and assure that the cassette recorder is properly connected to your system. To begin transfer of the text file to the audio cassette, hold down the control button (marked CTRL), and strike the character "T". Before striking the carriage return, put your cassette recorder in record mode and start the tape. Then strike the carriage return. Before recording, you should use the S command to find the begining and end of your text area. Make a note of it on the tape. The editor will then transfer control to the audio cassette routines. When the tape has been properly recorded, control will return to the Editor.

Reloading the Text from Audio Cassette

When you wish to place the text you previously stored on cassette back into memory for further editing or assembly, first prepare your cassette system for playback. Reload the cassette using the monitor tape load routine and entering the starting and ending address noted when you recorded the tape. When the tape has been read in, start the editor, give the starting address of the text in response to the BASE= query and answer "0" to the N OR 0? query. The text file is now ready for further processing.

The Query Command

Typing a Q command to the editor returns you to the BASE= query of the editor. You may then specify a new origin for further text entry of other text files. Previous text files will not be disturbed unless the files overlap in memory.

Changing The Prompt Character

It is sometimes convenient to have the editor type a prompt character at the beginning of each line when it is ready for input from the user. When the editor is initially entered, this prompt character is set to a null. If you wish a prompt character you simply type a question mark and carriage return and the editor will begin each line of input with a question mark as a prompt. If you wish to use a prompt character other than the question mark, type a question mark, the character you wish to use as a prompt, and a carriage return. For instance, typing a question mark, asterisk, carriage return will set the prompt character to an asterisk. To "turn off" the prompt character, simply type a question mark followed by a null (generated by depressing the control and shift buttons on your keyboard and striking the P key).

Using The ARESCO Assembler/Editor With An APPLE Printer

If you have a printer connected to your Apple II through an Apple Parallel Printer interface card, you may use your printer to print editor or assembler output. Before entering the editor type:

$1P^C$	(turn on the printer board)
I^C40N	(initialize output)
I^CI	(turn screen back on)

Then type 3743G to start the Editor. Type P A when you are in command mode to list your source program or A to print the assembled output. When you have returned to monitor mode, type $0P^C$ to turn the printer off.

THE ASSEMBLER

Table Of Contents

INTRODUCTION

The process of translating a mnemonic or symbolic form of a computer program to actual machine code is called an assembly, and a program which performs the translation is an assembler. The symbols used and rules of association for those symbols are the assembly language. In general, one assembly language statement will translate into one machine instruction. This distinguishes an assembler from a compiler, which may produce many machine instructions from a single statement.

Normally, digital computers use the binary number system for representation of data and instructions. Computers understand only ones and zeroes, corresponding to an "on" or "off" state. Human users, on the other hand, find it difficult to work with the binary number system and hence use a more convenient representation such as octal (base 8), decimal (base 10), or hexadecimal (base 16). Two representations of the 6502 operation to "load" information into an "accumulator" are shown here:

```
10101001          (binary)
   A9             (hexadecimal)
```

An instruction to move the value 21 (decimal) into the accumulator is:

```
A9 15             (hexadecimal)
```

Users still find numeric representations of instructions tedious to work with, and hence have developed symbolic representations. For example, the preceeding instruction might be written in assembly language as:

```
LDA #21           (assembly language)
```

In this case, LDA is a symbol for A9, LoaD the Accumulator. A computer program used to translate the symbolic form LDA to the numeric form A9 is called an assembler. The symbolic program is referred to as source code and the numeric program is the

object code. Only object code can be executed on the processor.

Each machine instruction to be executed has a symbolic name referred to as an opcode (operation code). The opcode for "store the contents of the accumulator" is STA. The opcode for "transfer the contents of the accumulator to index X" is TAX. There are 55 opcodes for the MOS 6502 processor (listed in Table 2). A machine instruction in assembly language consists of an opcode and (perhaps) operands, which specify the data on which the operation is to be performed.

Instructions may be labelled for reference by other instructions as shown in:

```
L2    LDA    #21
```

The label is L2, the opcode is LDA, and the operand is #21. At least one blank must separate the three parts (fields) of the instruction. Additional blanks may be inserted for ease of reading. Instructions for the ARESCO assembler have at most one operand, and many instructions have none. In these cases, the operation to be performed is completely specified by the opcode, as in CLC (clear the Carry bit).

Programming in assembly language requires learning the instruction set (opcodes), addressing conventions for referencing data, the data structures within the processor, as well as the structure of assembly language programs.

TABLE 2

6502 Instruction Set - Opcodes

ADC	Add with Carry to Accumulator
AND	"AND" to Accumulator
ASL	Shift Left One Bit (Memory or Accumulator)
BCC	Branch on Carry Clear
BCS	Branch on Carry Set
BEQ	Branch on Zero Result
BIT	Test Bits in Memory with Accumulator
BMI	Branch on Results Minus
BNE	Branch on Result not Zero
BPL	Branch on Result Plus
BRK	Force an Interrupt or Break
BVC	Branch on Overflow Clear
BVS	Branch on Overflow Set
CLC	Clear Carry Flag
CLD	Clear Decimal Mode
CLI	Clear Interrupt Disable Bit
CLV	Clear Overflow Flag
CMP	Compare Memory and Accumulator
CPX	Compare Memory and Index X
CPY	Compare Memory and Index Y
DEC	Decrement Memory by One
DEX	Decrement Index X by One
DEY	Decrement Index Y by One
EØR	Exclusive-or Memory with Accumulator
INC	Increment Memory by One
INX	Increment X by One
INY	Increment Y by One
JMP	Jump to New Location
JSR	Jump to New Location Saving Return Address
LDA	Transfer Memory to Accumulator
LDX	Transfer Memory to Index X
LDY	Transfer Memory to Index Y
LSR	Shift One Bit Right (Memory or Accumulator)
NØP	Do Nothing - No Operation
ØRA	"OR" Memory with Accumulator
PHA	Push Accumulator on Stack
PHP	Push Processor Status on Stack
PLA	Pull Accumulator from Stack
PLP	Pull Processor Status from Stack
RØL	Rotate One Bit Left (Memory or Accumulator)
RØR	Rotate One Bit Right (Memory or Accumulator)
RTI	Return From Interrupt
RTS	Return From Subroutine
SBC	Subtract Memory and Carry from Accumulator
SEC	Set Carry Flag
SED	Set Decimal Mode
SEI	Set Interrupt Disable Status
STA	Store Accumulator in Memory
STX	Store Index X in Memory
STY	Store Index Y in Memory
TAX	Transfer Accumulator to Index X
TAY	Transfer Accumulator to Index Y
TSX	Transfer Stack Register to Index X
TXA	Transfer Index X to Accumulator
TXS	Transfer Index X to Stack Register
TYA	Transfer Index Y to Accumulator